$$a = 16 \cdot (P(-1,7) - P(7,-1)) \quad (8\text{-}35)$$

$$b = (17*H+16)>>5) \quad (8\text{-}36)$$

$$c = (17*V+16)>>5) \quad (8\text{-}37)$$

and H and V are defined as:

$$H = \sum_{i=1}^{4} i \cdot (P(3+i,-1) - P(3-i,-1)) \quad (8\text{-}38)$$

$$V = \sum_{j=1}^{4} j \cdot (P(-1,3+j) - P(-1,3-j)) \quad (8\text{-}39)$$

8.6 Transform coefficient decoding and picture construction prior to deblocking

This subclause defines aspects related to transform coefficient decoding.

8.6.1 Zig-zag scan

The decoder maps the sequence of transform coefficient levels to the transform coefficient level positions. For this mapping, the scanning pattern is shown in Figure 8-12.

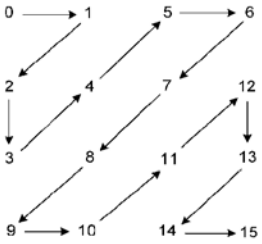

Figure 8-12 – Zig-zag scan

In the case of 16x16 intra macroblocks, the coefficients of the 4x4 luma DC transform are scanned in the same scan order as ordinary 4x4 coefficient blocks. Then for each 4x4 block of luma coefficients with AC coefficients to scan, the 15 remaining coefficients are scanned by starting the zig-zag scan at its second position.

The coefficients of the 2x2 chroma DC transform are scanned in raster order. Then for each 4x4 block of chroma coefficients with AC coefficients to scan, the 15 remaining coefficients are scanned by starting the zig-zag scan at its second position.

If adaptive_block_size_transform_flag == 1, 4x4, 4x8, 8x4, and 8x8 luma coefficient blocks are scanned using zig-zag scans and field scans as specified in subclause 12.~~3~~4.2.

8.6.2 Scaling and transformation

There are 52 different values of QP values that are used, ranging from 0 to 51, inclusive. The value of QP_C for chroma is determined from the current value of QP_Y. The scaling equations are defined such that the equivalent scaling parameter doubles for every increment of 6 in QP. Thus, there is an increase in scaling magnitude of approximately 12% from one QP to the next.

The value of QP_C shall be determined from the value of QP_Y as specified in Table 8-2:

Table 8-2 – Specification of QP_C as a function of QP_Y

QP_Y	<30	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
QP_C	$=QP_Y$	29	30	31	32	32	33	34	34	35	35	36	36	37	37	37	38	38	38	39	39	39	39

QP_Y shall be used as the QP to be applied for luma scaling and QP_C shall be used for chroma scaling.

The coefficients $R_{ij}^{(m)}$ defined in Equation 8-40 are used in Equations 8-43, 8-44, 8-46, 8-47 and 8-48.

$$R_{ij}^{(m)} = \begin{cases} V_{m0} & \text{for} \quad (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ V_{m1} & \text{for} \quad (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ V_{m2} & \text{otherwise;} \end{cases} \tag{8-40}$$

where the first and second subscripts of V are row and column indices, respectively, of the matrix defined as:

$$V = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \tag{8-41}$$

8.6.2.1 Luma DC coefficients in Intra 16x16 macroblock

After decoding the coefficient levels for a 4x4 block of luma DC coefficients coded in 16x16 intra mode and assembling these into a 4x4 matrix C of elements c_{ij}, a transform process shall be applied in a manner mathematically equivalent to the following process. The process uses application of a transform before the scaling process.

The transform for the 4x4 luma DC coefficients in 16x16 intra macroblocks is defined by:

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{8-42}$$

A bitstream conforming to this Recommendation | International Standard shall not contain indicated data that results in any element of F that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

After the transform, scaling is performed according to the following:

a) If QP is greater than or equal to 12, then the scaled result shall be calculated as

$$DC_{ij} = [F_{ij} \cdot R_{00}^{(QP\%6)}] << (QP/6 - 2), \quad i,j = 0,\ldots,3. \tag{8-43}$$

b) If QP is less than 12, then the scaled results shall be calculated as

$$DC_{ij} = [F_{ij} \cdot R_{00}^{(QP\%6)} + 2^{1-QP/6}] >> (2 - QP/6), \quad i,j = 0,\ldots,3. \tag{8-44}$$

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in element of DC_{ij} that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

8.6.2.2 Chroma DC coefficients

After decoding the coefficient levels for a 2x2 block of chroma DC coefficients and assembling these into a 2x2 matrix C of elements c_{ij}, the transform process is applied before the scaling process.

Definition of transform:

$$F = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (8\text{-}45)$$

A bitstream conforming to this Recommendation | International Standard shall not contain indicated data that results in any element of F that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

After the transform, scaling is performed according to the following.

a) If QP is greater than or equal to 6, then the scaling result shall be calculated as

$$DC_{ij} = [F_{ij} \cdot R_{00}^{(QP\%6)}] << (QP/6 - 1), \quad i, j = 0,...,3. \quad (8\text{-}46)$$

b) If QP is less than 6, then the scaling results shall be calculated by

$$DC_{ij} = [F_{ij} \cdot R_{00}^{(QP\%6)}] >> 1, \quad i, j = 0,...,3. \quad (8\text{-}47)$$

A bitstream conforming to this Recommendation | International Standard shall not contain indicated data that results in any element of DC_{ij} that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

8.6.2.3 Residual 4x4 blocks

Scaling of 4x4 block coefficient levels c_{ij} other than those as specified in subclauses 8.6.2.1 and 8.6.2.2 shall be performed according to Equation 8-48

$$w_{ij} = [c_{ij} \cdot R_{ij}^{(QP\%6)}] << (QP/6), \quad i, j = 0,...,3. \quad (8\text{-}48)$$

A bitstream conforming to this Recommendation | International Standard shall not contain indicated data that results in a value of w_{ij} that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

After constructing an entire 4x4 block of scaled transform coefficients and assembling these into a 4x4 matrix W of elements w_{ij} illustrated as

$$W = \begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} \quad (8\text{-}49)$$

in which the w_{00} element may be a result DC_{ij} from Equation 8-43, 8-44, 8-46, or 8-47; or may be from Equation 8-48, as appropriate, the transform process shall convert the block of reconstructed transform coefficients to a block of output samples in a manner mathematically equivalent to the following process:

a) First, each row of reconstructed transform coefficients is transformed using a one-dimensional transform, and

b) Second, each column of the resulting matrix is transformed using the same one-dimensional transform.

The one-dimensional transform is defined as follows for four input samples w_0, w_1, w_2, w_3.

a) First, a set of intermediate values is computed:

$$z_0 = w_0 + w_2 \quad (8\text{-}50)$$

$$z_1 = w_0 - w_2 \quad (8\text{-}51)$$

$$z_2 = (w_1 >> 1) - w_3 \quad (8\text{-}52)$$

$$z_3 = w_1 + (w_3 >> 1) \quad (8\text{-}53)$$

b) Then the transformed result is computed from these intermediate values

$$x_0 = z_0 + z_3 \tag{8-54}$$

$$x_1 = z_1 + z_2 \tag{8-55}$$

$$x_2 = z_1 - z_2 \tag{8-56}$$

$$x_3 = z_0 - z_3 \tag{8-57}$$

A bitstream conforming to this Recommendation | International Standard shall not contain indicated data that results in a value of z_0, z_1, z_2, z_3, x_0, x_1, x_2, or x_3 that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive, in either the first (horizontal) or second (vertical) stage of application of this transformation process. A bitstream conforming to this Recommendation | International Standard shall not contain indicated data that results in a value of x_0, x_1, x_2, or x_3 that exceeds the range of integer values from -2^{15} to $2^{15}-33$, inclusive, in the second (vertical) stage of application of this transformation process.

After performing the transform in both the horizontal and vertical directions to produce a block of transformed samples,

$$X' = \begin{bmatrix} x'_{00} & x'_{01} & x'_{02} & x'_{03} \\ x'_{10} & x'_{11} & x'_{12} & x'_{13} \\ x'_{20} & x'_{21} & x'_{22} & x'_{23} \\ x'_{30} & x'_{31} & x'_{32} & x'_{33} \end{bmatrix}, \tag{8-58}$$

the final reconstructed sample residual values shall be obtained as

$$X''_{ij} = [x'_{ij} + 2^5] >> 6 \tag{8-59}$$

If adaptive_block_size_transform_flag == 1, scaling and inverse transform for 4x8, 8x4, and 8x8 coefficient blocks is specified in subclause 12.~~3~~4.3.

8.6.3 Adding decoded samples to prediction with clipping

Finally, the reconstructed sample residual values X'' from Equation 8-59 are added to the prediction values P_{ij} from motion compensated prediction or spatial prediction and clipped to the range of 0 to 255 to form the final decoded sample result prior to application of the deblocking filter:

$$S'_{ij} = \text{Clip1}(P_{ij} + X''_{ij}) \tag{8-60}$$

8.7 Deblocking Filter

A conditional filtering shall be applied to all macroblocks of a picture. This filtering is done on a macroblock basis, with macroblocks being processed in raster-scan order throughout the picture. For luma, as the first step, the 16 samples of the 4 vertical edges of the 4x4 raster shall be filtered beginning with the left edge, as shown in Figure 8-13. Filtering of the 4 horizontal edges (vertical filtering) follows in the same manner, beginning with the top edge. The same ordering applies for chroma filtering, with the exception that 2 edges of 8 samples each are filtered in each direction. This process also affects the boundaries of the already reconstructed macroblocks above and to the left of the current macroblock. Picture edges are not filtered.

When mb_adaptive_frame_field_flag = 1, a MB may be coded in frame or field decoding mode. For frame MB, deblocking is performed on the frame samples. In this case, if neighbouring MB pairs are field MBs, they shall be converted into frame MB pairs (Figure 8-3) before deblocking. For field MB, deblocking is performed on the field samples of the same field parity. In this case, if neighbouring MB pairs are frame MBs, they shall be converted into field MB pairs (Figure 8-3) before deblocking.

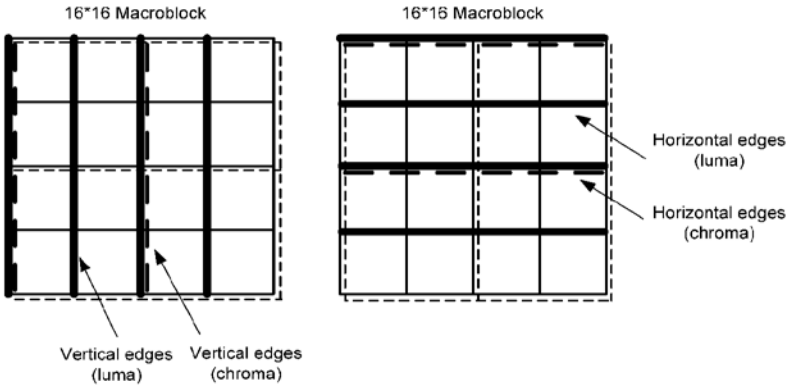

Figure 8-13 – Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dotted lines)

Intra prediction of a macroblock shall be done using the unfiltered content of the already decoded neighbouring macroblocks. Depending on the implementation, the values necessary for intra prediction may need to be stored before filtering in order to be used in the intra prediction of the macroblocks to the right and below the current macroblock.

When pic_structure indicates a field picture all decoding operations for the deblocking filter are based solely on samples within the current field.

8.7.1 Content dependent boundary filtering strength

For each boundary between neighbouring 4x4 luma blocks, a "Boundary Strength" Bs is assigned as shown in Figure 8-14. If Bs=0, filtering is skipped for that particular edge. In all other cases filtering is dependent on the local sample properties and the value of Bs for this particular boundary segment.

For each edge, if one of the neighbouring blocks is intra-coded, a relatively strong filtering (Bs=3) is applied. A special procedure with even stronger filtering might be applied on intra-coded macroblock boundaries (Bs=4). If neither of the blocks are intra-coded and at least one of them contains non-zero coefficients, medium filtering strength (Bs=2) is used. If none of the previous conditions are satisfied, filtering takes place with Bs=1 if at least one of the following conditions is satisfied: (a) prediction of the two blocks is formed using different reference frames or a different number of reference frames. (b) a pair of motion vectors from the two blocks is referencing the same frame and either component of this pair has a difference of more than one sample. Otherwise filtering is skipped for that particular edge (Bs=0).

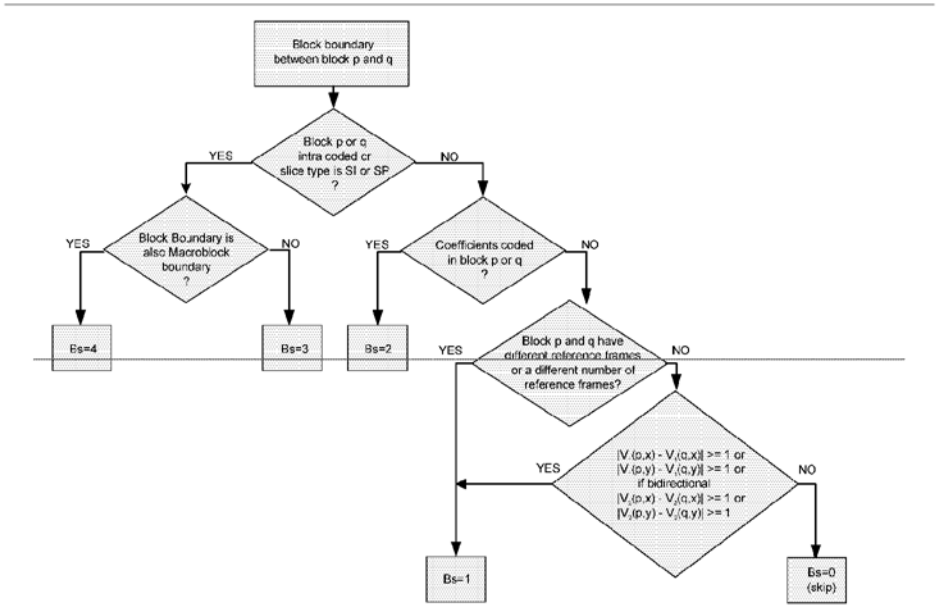

Figure 8-14 – Flow chart for determining the boundary strength (Bs), for the block boundary between two neighbouring blocks p and q, where V_1(~~r~~p,x), V_1(p,y) and V_2(~~r~~p, x), V_2(p, y) are the horizontal and vertical components of the motion vectors of block p for ~~to~~ the first and second reference frames or fields.

8.7.2 Thresholds for each block boundary

Figure 8-15 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

In the following description, the set of eight samples across a 4x4 block horizontal or vertical boundary is denoted as shown in Figure 8-15 with the actual boundary lying between p_0 and q_0. Uppercase letters indicate filtered samples and lower case letters indicate unfiltered samples with regard to the current edge filtering operation. However, p_1 and p_2 may indicate samples that have been modified by the filtering of a previous block edge.

Sets of samples across this edge are only filtered if the condition

$$Bs \neq 0 \;\&\&\; |p_0 - q_0| < \alpha \;\&\&\; |p_1 - p_0| < \beta \;\&\&\; |q_1 - q_0| < \beta \qquad (8\text{-}61)$$

is true. The values of the thresholds α and β are dependent on the average value of QP for the two blocks as well as on a pair of index offsets "Filter_Offset_A" and "Filter_Offset_B" that may be transmitted in the slice header for the purpose of modifying the characteristics of the filter. The average QP value for the two blocks is computed as $QP_{av} = (QP_p+QP_q)>>1$. The index used to access the α-table (Table 8-3), as well as the C0-table (Table 8-4) that is used in the default filter mode, is computed as:

$$Index_A = Clip3(0, 51, QP_{av} + Filter\_Offset\_A) \qquad (8\text{-}62)$$

NOTE - In SP and SI slices, QP_{av} is calculated in the same way as in other slice types. QS_Y from Equation 7-8 is not used in the deblocking filter.

The index used to access the β-table (Table 8-3) is computed as:

$$Index_B = Clip3(0, 51, QP_{av} + Filter\_Offset\_B) \qquad (8\text{-}63)$$

If adaptive_block_size_transform_flag == 1, $Index_A$ and $Index_B$ are calculated as specified in subclause 12.34.4.

The relationships between the indices (Equations 8-62 and 8-63) and the thresholds (α and β) are shown in Table 8-3.

Table 8-3 – QP_{av} and offset dependent threshold parameters α and β

	$Index_A$ (for α) or $Index_B$ (for β)																								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
α	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	6	7	8	9	10	12	13
β	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	3	3	3	4	4	4

Table 8-3 (concluded)

	$Index_A$ (for α) or $Index_B$ (for β)																								
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
α	15	17	20	22	25	28	32	36	40	45	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β	6	6	7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

8.7.3 Filtering of edges with Bs < 4

Two types of filtering are defined. In the default case with 0 < B < 4, Equations 8-64, 8-65 and 8-66 are used to filter p_0 and q_0:

$$\Delta = Clip3(-C, C, (((q_0 - p_0) << 2 + (p_1 - q_1) + 4) >> 3)) \qquad (8\text{-}64)$$

Formatted: Subscript
Formatted: Subscript
Formatted: Subscript

$$P_0 = \text{Clip1}(p_0 + \Delta) \quad (8\text{-}65)$$

$$Q_0 = \text{Clip1}(q_0 - \Delta) \quad (8\text{-}66)$$

where C is determined as specified below.

The two intermediate threshold variables

$$a_p = |p_2 - p_0| \quad (8\text{-}67)$$

$$a_q = |q_2 - q_0| \quad (8\text{-}68)$$

shall be used to determine whether filtering for the luma samples p_1 and q_1 is taking place at this position of the edge.

If $a_p < \beta$ for a luma edge, a filtered sample P_1 shall be produced as specified by

$$P_1 = p_1 + \text{Clip3}(-C0, C0, (p_2 + ((p_0 + q_0) >> 1) - (p_1 << 1)) >> 1) \quad (8\text{-}69)$$

If $a_q < \beta$ for a luma edge, a filtered sample Q_1 shall be produced as specified by

$$Q_1 = q_1 + \text{Clip3}(-C0, C0, (q_2 + ((p_0 + q_0) >> 1) - (q_1 << 1)) >> 1) \quad (8\text{-}70)$$

where C0 is specified in Table 8-4. Chroma samples p_1 and q_1 are never filtered.

C is determined by setting it equal to C0 and then incrementing it by one if $a_p < \beta$, and again by one if $a_q < \beta$.

Table 8-4 – Value of filter clipping parameter C0 as a function of Index_A and Bs

	Index_A																								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Bs = 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
Bs = 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
Bs = 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Table 8-4 (concluded)

	Index_A																								
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Bs = 1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13
Bs = 2	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17
Bs = 3	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25

8.7.4 Filtering of edges with Bs = 4

When Bs is equal to 4 and the following condition holds:

$$a_p < \beta \;\&\&\; |p_0 - q_0| < ((\alpha >> 2) + 2) \quad (8\text{-}71)$$

filtering of the left/upper side of the block edge is specified by Equations 8-72 and 8-73.

$$P_0 = (p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4) >> 3 \quad (8\text{-}72)$$

$$P_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2 \quad (8\text{-}73)$$

In the case of luma filtering, the filter in Equation 8-74 is also applied.

$$P_2 = (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) >> 3 \quad (8\text{-}74)$$

Otherwise, if the condition of 8-71 does not hold, the filter in Equation 8-75 is applied.

$$P_0 = (2*p_1 + p_0 + q_1 + 2) >> 2 \quad (8\text{-}75)$$

Similarly, for filtering of the right/lower side of the edge, if the following condition holds:

$$a_q < \beta \;\&\&\; |p_0 - q_0| < ((\alpha >> 2) + 2) \quad (8\text{-}76)$$

filtering is defined by Equations 8-77 and 8-78.

$$Q_0 = (p_1 + 2*p_0 + 2*q_0 + 2*q_1 + q_2 + 4) >> 3 \quad (8\text{-}77)$$

$$Q_1 = (p_0 + q_0 + q_1 + q_2 + 2) >> 2 \quad (8\text{-}78)$$

In the case of luma filtering, the filter in Equation 8-~~74~~ 79 is also applied.

$$Q_2 = (2*q_3 + 3*q_2 + q_1 + q_0 + p_0 + 4) >> 3 \quad (8\text{-}79)$$

Otherwise, if the condition of 8-76 does not hold, the ~~following~~ filter in Equation 8-80 is applied:

$$Q_0 = (2*q_1 + q_0 + p_1 + 2) >> 2 \quad (8\text{-}80)$$

9 Entropy Coding

9.1 Variable Length Coding

9.1.1 Exp-Golomb entropy coding

The table of Exp-Golomb codewords is written in the following compressed form.

```
                1
              0 1 x0
            0 0 1 x1 x0
          0 0 0 1 x2 x1 x0
        0 0 0 0 1 x3 x2 x1 x0
        ...................
```

where xn take values 0 or 1. A codeword can be referred by its length in bits (L = 2n-1) and INFO = xn,…x1, x0. Notice that the number of bits in INFO is n-1 bits. The codewords are numbered from 0 and upwards. The definition of the numbering is:

Code_num = $2^{L/2}$ + INFO -1 (L/2 denotes division with truncation and INFO = 0 when L = 1). The first 10 code numbers and codewords are specified explicitly in Table 9-1. As an example, for the code number 5, L = 5 and INFO = 10 (binary) = 2 (decimal).

Table 9-1 – Code number and Exp-Golomb codewords in explicit form and used as ue(v)

Code_num	Code word
0	1
1	0 1 0
2	0 1 1
3	0 0 1 0 0
4	0 0 1 0 1

5	0 0 1 1 0
6	0 0 1 1 1
7	0 0 0 1 0 0 0
8	0 0 0 1 0 0 1
9	0 0 0 1 0 1 0

When L (L = 2N-1) and INFO is known, the regular structure of the table makes it possible to create a codeword usning the structure of the table. A decoder shall decode a codeword by reading in N bit prefix followed by N-1 INFO. L and INFO are then available. For each parameter to be coded, there is a conversion rule from the parameter value to the code number (or L and INFO).

9.1.2 Unsigned Exp-Golomb entropy coding

The value of syntax elements that are represented by unsigned Exp-Golomb entropy coding directly corresponds to the code_num value of Table 9-1. This type of entropy coding is indicated via ue(v).

9.1.3 Signed Exp-Golomb entropy coding

The syntax elements that are represented by signed Exp-Golomb entropy coding are assigned to the code_num by ordering using their absolute values in increasing order and representing the positive value with the lower code_num. Table 9-2 provides the assignment rule.

Table 9-2 – Assignment of symbol values and code_nums for signed Exp-Golomb entropy coding se(v)

Code number	Symbol value
0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3
7	4
8	-4
9	5
10	-5
k	$(-1)^{k+1}$ Ceil(k÷2)

This type of entropy coding is denoted as se(v).

9.1.4 Mapped Exp-Golomb entropy coding

Table 9-3 specifies the assignment of all mapped Exp-Golomb-coded slice data symbols. This type of entropy coding is indicated via me(v). These symbols are decoded differently when entropy_coding_mode = = 1.

If adaptive_block_size_transform_flag = = 1, additional syntax elements are mapped to the Exp-Golomb code as specified in Table 12-65.

Table 9-3 – Assignment of codeword number and parameter values for mapped Exp-Golomb-coded symbols

Code number	coded_block_pattern assignment to macroblock prediction types		Tcoeff_chroma_DC1		Tcoeff_chroma_AC1 Tcoeff_luma1 Zig-zag scan	
	Intra, SIntra	Pred, SPred	Level	Run	Level	Run
0	47	0	EOB	-	EOB	-
1	31	16	1	0	1	0
2	15	1	-1	0	-1	0
3	0	2	2	0	1	1
4	23	4	-2	0	-1	1
5	27	8	1	1	1	2
6	29	32	-1	1	-1	2
7	30	3	3	0	2	0
8	7	5	-3	0	-2	0
9	11	10	2	1	1	3
10	13	12	-2	1	-1	3
11	14	15	1	2	1	4
12	39	47	-1	2	-1	4
13	43	7	1	3	1	5
14	45	11	-1	3	-1	5
15	46	13	4	0	3	0
16	16	14	-4	0	-3	0
17	3	6	3	1	2	1
18	5	9	-3	1	-2	1
19	10	31	2	2	2	2
20	12	35	-2	2	-2	2
21	19	37	2	3	1	6
22	21	42	-2	3	-1	6
23	26	44	5	0	1	7
24	28	33	-5	0	-1	7
25	35	34	4	1	1	8
26	37	36	-4	1	-1	8
27	42	40	3	2	1	9
28	44	39	-3	2	-1	9
29	1	43	3	3	4	0
30	2	45	-3	3	-4	0
31	4	46	6	0	5	0

32	8	17	-6	0	-5	0
33	17	18	5	1	3	1
34	18	20	-5	1	-3	1
35	20	24	4	2	3	2
36	24	19	-4	2	-3	2
37	6	21	4	3	2	3
38	9	26	-4	3	-2	3
39	22	28	7	0	2	4
40	25	23	-7	0	-2	4
41	32	27	6	1	2	5
42	33	29	-6	1	-2	5
43	34	30	5	2	2	6
44	36	22	-5	2	-2	6
45	40	25	5	3	2	7
46	38	38	-5	3	-2	7
47	41	41	8	0	2	8
K	-	-	see below	see below	see below	see below

For the entries above the horizontal line, the table is needed for relation between code number and Level/Run/EOB. For the remaining Level/Run combination there is a simple rule. The Level/Run combinations are assigned a code number according to the following priority: 1) sign of Level (+ -) 2) Run (ascending) 3) absolute value of Level (ascending).

9.1.5 Entropy coding for Intra

In intra mode, prediction is always used for each sub block in a macroblock.

9.1.5.1 Coding of Intra 4x4 and SIntra 4x4 prediction modes

The chosen intra-prediction mode (intra_pred_mode) of a 4x4 block is highly correlated with the prediction modes of adjacent blocks. This is illustrated in Figure 9-1. When the prediction modes of A and B are known (including the case that A or B or both are outside the slice) the most probable mode (most_probable_mode) of C is given. If one of the blocks A or B is "outside" the most probable mode is equal to prediction mode 2. Otherwise it is equal to the minimum of modes used for blocks A and B. When an adjacent block is coded by 16x16 intra mode, prediction mode is "mode 2: DC prediction"; when it is coded a non-intra macroblock, prediction mode is "mode 2: DC prediction" in the usual case and "outside" in the case of constrained intra update.

To signal prediction mode number for a 4x4 block first parameter use_most_probable_mode is transmitted. This parameter is represented by 1 bit codeword and can take values 0 or 1. If use_most_probable_mode is equal to 1 the most probable mode is used. Otherwise an additional parameter remaining_mode_selector, which can take value from 0 to 7 is sent as 3 bit codeword. The codeword is a binary representation of remaining_mode_selector value. The prediction mode number is calculated as:

```
if (remaining_mode_selector < most_probable_mode)
        intra_pred_mode = remaining_mode_selector;
else
        intra_pred_mode = remaining_mode_selector+1;
```

The ordering of prediction modes assigned to blocks C is therefore the most probable mode followed by the remaining modes in the ascending order.

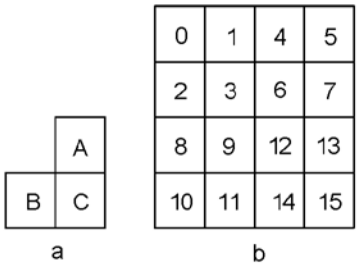

Figure 9-1 – a) Prediction mode of block C to be established, where A and B are adjacent blocks. b) order of intra prediction information in the bitstream

9.1.5.2 Coding of mode information for Intra-16x16 mode

Three numbers are specified at the end of the names of Intra-16x16 modes as defined in Table 7-10 as a function of mb_type. The first of these numbers is termed *Imode* and ranges from 0 to 3, inclusive. The second is termed *nc* and contains the coded_block_pattern bits for chroma as specified in subclause 9.2.1.6. The third and final of these numbers is termed *ac_flag*. *ac_flag* equal to zero indicates that there are no AC coefficients in the 16x16 block. *ac_flag* equal to 1 indicates that at least one AC coefficient is present for the 16x16 block, requiring scanning of AC coefficient values for all 16 of the 4x4 blocks in the 16x16 block.

9.1.6 Context-based adaptive variable length coding (CAVLC) of transform coefficients

CAVLC (Context-based Adaptive VLC) is the method used for decoding of transform coefficients. The following coding elements are used:

1. If there are non-zero coefficients, it is typically observed that there is a string of coefficients at the highest frequencies that are +1. The coding element coeff_token gives the total number of coefficients (from now referred to as TotalCoeffs) and also contains the number of "Trailing 1s" (from now referred to as T1s). ~~A common parameter called TotalCoeffs is used that contains the number of coefficients as well as the number of "Trailing 1s" (from now referred to as T1s). For T1s only the sign has to be decoded.~~

2. For T1s the sign is decoded from trailing_ones_sign and the level magnitude is 1.

3. For coefficients other than the T1s, level information is decoded from coeff_level.

~~3.~~4. The Run information is decoded. Since the number of coefficients is already known, this limits possible values for Run. Run is split into the Total number of zeros before all coefficients and Run before each non-zero coefficient, given by total_zeros and run_before.

Zig-zag scanning as described in subclause 9.4.1 is used, but in the decoding of coefficient data, both levels and runs, the scanning is done in reverse order. Therefore~~,~~ ~~in the Level information,~~ the signs of T1s are decoded first (in reverse order), then the Level information of the last coefficient in the zig-zag scan order not included in the T1s~~where this is needed~~, and so on. Run information is decoded similarly. First Total number of zeros in Runs is decoded, followed by Run before the last nonzero coefficient in the zig-zag scan order, and so on.

If adaptive_block_size_transform_flag == 1, the VLC method for decoding 4x4, 4x8, 8x4, and 8x8 luma coefficient blocks is specified in subclause 12.~~4.2~~5.1.

9.1.6.1 Entropy decoding of the number of coefficients and trailing ones: coeff_token

The syntax element coeff_token is decoded using the VLC specified in Tables 9-4 to 9-7.

Four VLC tables are used for combined decoding of number of coefficients and T1s, i.e. one codeword signals both parameters. VLCs are listed in the tables below. T1s is clipped to 3. Any remaining trailing 1s are decoded as normal levels. The variable ~~Num~~TotalCoeff ~~=~~is the value returned by the function ~~num~~total_coeff( ); the variable T1s is the value returned by the function trailing_ones( ); and~~,~~ the variable NumCoeff related to these quantities by TotalCoeff-T1s ~~= trailing_ones( )~~.

Formatted: Heading 4,Heading 4 Char,Heading 4 Char1 Char,Heading 4 Char Char Char,h4,H4,H41,Titre 4

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Formatted: Bullets and Numbering

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Table 9-4 – coeff_token: ~~num~~total_coeff~~s~~( ) / trailing_ones( ): Num-VLC0

trailing_ones( )	0	1	2	3
~~num~~total_coeff( )				
0	1	-	-	-
1	000101	01	-	-
2	00000111	000100	001	-
3	000000111	00000110	0000101	00011
4	0000000111	000000110	00000101	000011
5	00000000111	0000000110	000000101	0000100
6	0000000001111	00000000110	0000000101	00000100
7	0000000001011	0000000001110	00000000101	000000100
8	0000000001000	0000000001010	0000000001101	0000000100
9	00000000001111	00000000001110	0000000001001	00000000100
10	00000000001011	00000000001010	00000000001101	0000000001100
11	000000000001111	000000000001110	00000000001001	00000000001100
12	000000000001011	000000000001010	000000000001101	00000000001000
13	0000000000001111	000000000000001	000000000001001	000000000001100
14	0000000000001011	0000000000001110	0000000000001101	000000000001000
15	0000000000000111	0000000000001010	0000000000001001	0000000000001100
16	0000000000000100	0000000000000110	0000000000000101	0000000000001000

Table 9-5 – coeff_token: ~~num_coeff~~total_coeff( ) / trailing_ones( ): Num-VLC1

trailing_ones( )	0	1	2	3
~~num_coeff~~total_coeff( )				
0	11	-	-	-
1	001011	10	-	-
2	000111	00111	011	-
3	0000111	001010	001001	0101
4	00000111	000110	000101	0100
5	00000100	0000110	0000101	00110
6	000000111	00000110	00000101	001000
7	000000001111	000000110	000000101	000100
8	000000001011	000000001110	000000001101	0000100
9	000000001111	000000001010	000000001001	000000100
10	000000001011	000000001110	000000001101	000000001100
11	000000001000	000000001010	000000001001	000000001000

12	0000000001111	0000000001110	0000000001101	000000001100
13	0000000001011	0000000001010	0000000001001	0000000001100
14	0000000000111	00000000001011	0000000000110	0000000001000
15	00000000001001	00000000001000	00000000001010	0000000000001
16	00000000000111	00000000000110	00000000000101	00000000000100

Table 9-6 – coeff_token: ~~num_coeff~~total_coeff( ) / trailing_ones( ): Num-VLC2

trailing_ones( )	0	1	2	3
~~num_coeff~~total_coeff( )				
0	1111	-	-	-
1	001111	1110	-	-
2	001011	01111	1101	-
3	001000	01100	01110	1100
4	0001111	01010	01011	1011
5	0001011	01000	01001	1010
6	0001001	001110	001101	1001
7	0001000	001010	001001	1000
8	00001111	0001110	0001101	01101
9	00001011	00001110	0001010	001100
10	000001111	00001010	00001101	0001100
11	000001011	000001110	00001001	00001100
12	000001000	000001010	000001101	00001000
13	0000001101	000000111	000001001	000001100
14	0000001001	0000001100	0000001011	0000001010
15	0000000101	0000001000	0000000111	0000000110
16	0000000001	0000000100	0000000011	0000000010

Table 9-7 – coeff_token: ~~num_coeff~~total_coeff( ) / trailing_ones( ): Num-VLC_Chroma_DC

trailing_ones( )	0	1	2	3
~~num_coeff~~total_coeff( )				
0	01	-	-	-
1	000111	1	-	-
2	000100	000110	001	-
3	000011	0000011	0000010	000101

9.1.6.2 Table selection

For all elements, except chroma DC, a choice between three tables and one FLC is made. N is a value used for Table selection. Selection is done as follows: N is calculated based on the number of coefficients in the block above and to the left of the current block: N_U and N_L. In the table below, X means that the block is available in the same slice. The block's

coding mode is not taken into account when determining availability. When finding the block above and to the left for a block of Intra16x16 DC coefficients, the location of the block is assumed to be (0,0), i.e. the upper left corner of the macroblock.

Table 9-8 – Calculation of N for Num-VLCN

Upper block (N_U)	Left block (N_L)	N
X	X	$(N_L+N_U)/2$
X		N_U
	X	N_L
		0

0 <= N < 2 : Num-VLC0

2 <= N < 4 : Num-VLC1

4 <= N < 8 : Num-VLC2

N >= 8 : 6 bit FLC xxxxyy, as follows:

As a part of the coeff_token, ~~Num~~TotalCoeff−1 is transmitted in the first 4 bits (xxxx) and T1s is transmitted as the last 2 bits (yy). There is one exception: the codeword 000011 represents ~~Num~~TotalCoeff=0.

For chroma DC, Num-VLC_Chroma_DC is used.

9.1.6.3 Decoding of level information: coeff_level

First, the sign of T1s are decoded from 1 bit each of trailing_ones_sign. A maximum of 3 bits are read.

For the remaining level information, ~~four~~ seven structured VLCs are used to decode levels. The structured level tables are explained in Tables ~~10-11~~9-9 to 9-15. Lev-VLC0 has its own structure while the other tables, **Lev-VLC*N*, *N* = 1 to 6**, share a common structure.

Table 9-9 – Level tables

Lev-VLC0		
Code no	Code	~~Level Code~~LevelCode (±1, ±2..)
0	1	1
1	01	-1
2	001	2
3	0001	-2
..	..	..
13	00000000000001	-7
14-29	000000000000001xxxx	±8 to ±15
30->	0000000000000001xxxxxxxxxxxx	±16 ->

For **Lev-VLC*N*, *N* = 1 to 6**, the following structure is used:

let level_code be the level information to be decoded from the VLC tables,

if (|level_code|-1) < (15<<(*N*-1)),

Code: 0….01x..xs,

where number of 0's = (|level_code|-1) >> (*N* -1),

number of x's = *N*-1,

value of x's = (|level_code|-1) % $2^{(N-1)}$,

s = sign bit (0 – positive, 1 – negative)

else,

28-bit escape code: 0000 0000 0000 0001 xxxx xxxx xxxs,

where value of x's = (|level_code|-1) - (15<<(N -1)),

s = sign bit (0 – positive, 1 – negative)

Table 9-10 – Level VLC1

Lev-VLC1		
Code no	Code	~~Level Code~~LevelCode (±1, ±2..)
0-1	1s	±1
2-3	01s	±2
..	..	..
28-43	000000000000001s	±15
44 ->	0000000000000001xxxxxxxxxxxs	±16 ->

Table 9-11 – Level VLC2

Lev-VLC2		
Code no	Code	~~Level Code~~LevelCode (±1, ±2..)
0-3	1xs	±1 to ±2
4-7	01xs	±3 to ±4
..	..s	..
56-71	0000000000000001xs	±29 to ±30
72 ->	0000000000000001xxxxxxxxxxxs	±31 ->

Table 9-12 – Level VLC3

Lev-VLC3		
Code no	Code	~~Level Code~~LevelCode (±1, ±2..)
0-7	1xxs	±1 to ±4
8-16	01xxs	±5 to ±8
..	..	..
112-127	0000000000000001xxs	±57 to ±60
128 ->	0000000000000001xxxxxxxxxxxs	±61 ->

Formatted: Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next

Table 9-13 – Level VLC4

Lev-VLC4		
Code no	Code	~~Level Code~~LevelCode (±1, ±2..)
0-7	1xxxs	±1 to ±8~~±1 to ±4~~
8-16	01xxxs	±9 to ±16~~±5 to ±8~~
..	..	..
112-127	000000000000001xxxs	±113 to ±120~~±57 to ±60~~
128 ->	0000000000000001xxxxxxxxxxxs	±121 ->~~±61 ->~~

Table 9-14 – Level VLC5

Lev-VLC5		
Code no	Code	~~Level Code~~LevelCode (±1, ±2..)
0-7	1xxxxs	±1 to ±16~~±1 to ±4~~
8-16	01xxxxs	±17 to ±32~~±5 to ±8~~
..	..	..
112-127	000000000000001xxxxs	±225 to ±240~~±57 to ±60~~
128 ->	0000000000000001xxxxxxxxxxxxs	±241 ->~~±61 ->~~

Table 9-15 – Level VLC6

Lev-VLC6		
Code no	Code no	~~Level Code~~LevelCode (±1, ±2..)
0-15	1xxxxxs	±1 to ±32
8-16	01 xxxxxs	±33 to ±64
..	..	..
112-127	000000000000001xxxxxs	±449 to ±480
128 ->	0000000000000001xxxxxxxxxxs	±481 ->

Normally all coefficient levels (coeff_level) are equal to the decoded LevelCode~~level code~~ values given in Tables 9-9 to 9-15.~~;~~ ~~however~~However, when T1s is less than 3, the level of the first coefficient (after T1s) is equal to the decoded ~~level code~~LevelCode plus 1:

```
If (first coefficient && trailing_ones() < 3)
        coeff_level = (|level_code| + 1) * sign(level_code)
    else
        coeff_level = level_code
```

The last two entries in Lev-VLC0 table are escape codes. The first escape code with 19 bits, four "x"'s, is used to decode the 8 levels above the last regularly coded level. The next escape code with 28-bits, 12 "x"'s, is used to decode all remaining levels. For Lev-VLC1 to Lev-VLC6 tables, only the 28-bit escape code is used.

Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Left, Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: Keep with next
Formatted: enumlev1

9.1.6.3 Table selection

Selections of the tables are changed ~~along with~~during the decoding process based on number of coefficients, number of trailing ones, and the size of the previously decoded level value (coeff_level).

Let VLC denote the ~~*N* in Lev-VLC*N*~~ **Lev-VLC*N* (N=0-6) to be used**. After each level is decoded, the VLC ~~number~~N is updated according to the following method, where Level is the absolute value of the previously decoded level (coeff_level).

```
// VLC initialization for decoding first level
if (total_coeff(coeff_token) > 10 && trailing_ones(coeff_token)< 3)
   VLC = 1        // use Lev-VLC1 for first level
else
   VLC = 0        // use Lev-VLC0 for first level
// Assign FirstCoeff
// Decode level_code here and assign coeff_level and Level
// VLC update after decoding each level
vlc_inc table[7] = {0, 3, 6, 12, 24, 48, 2^15}
if (Level > vlc_inc[VLC])
   VLC ++
if (FirstCoeff~~first coefficient~~ and Level > 3)
   VLC = 2
```

The first coefficient is always decoded with Lev-VLC0 or Lev-VLC1 while the rest of the coefficients are always decoded with Lev-VLC1 to Lev-VLC6.

The same procedure is used for chroma AC and DC coefficient levels.

9.1.6.4 Decoding of run information

Run decoding is separated in total number of Zeros (i.e. the number of zeros located before the last non-zero coefficient in the zig-zag scan) and Run (of zeros) before each coefficient.

9.1.6.4.1 Entropy Decoding of the total number of zeros: total_zeros

~~The parameter TotalZeros is the sum of all zeros located before the last non-zero coefficient in a forward scan. For example, given the string of coefficients 0 0 3 0 0 4 0 0 0 0 2 0 1 0 0 0, TotalZeros will be 2+2+4+1=9. Since NumCoeff is already known, it determines the maximum possible value of TotalZeros. One out of 15 VLC tables is chosen based on NumCoeff.~~

~~If NumCoeff indicates that all coefficients are non-zero, TotalZeros is not decoded since it is known to be zero~~The variable TotalZeros as given by total_zeros, is the sum of all zeros located before the last non-zero coefficient in a zig-zag scan. For example, given the string of coefficients 0 0 3 0 0 4 0 0 0 0 2 0 1 0 0 0, TotalZeros will be 2+2+4+1=9. Since TotalCoeff is already known, it determines the maximum possible value of TotalZeros. One out of 15 VLC tables is chosen based on TotalCoeff.

If TotalCoeff indicates that all coefficients are non-zero, TotalZeros is not decoded since it is known to be zero

Table 9-16 – total_zeros~~TotalZeros~~ tables for all 4x4 blocks

TotalNumCoeff *TotalZeros*	1	2	3	4	5	6	7	
0	1	111	0101	00011	0101	000001	000001	
1	011	110	111	111	0100	00001	00001	
2	010	101	110	0101	0011	111	101	
3	0011	100	101	0100	111	110	100	

Formatted: Font: (Default) Courier New

Formatted: enumlev1

Formatted: Font: (Default) Courier New

Formatted: Font: (Default) Courier New

Formatted: Font: Not Bold

4	0010	011	0100	110	110	101	011	
5	00011	0101	0011	101	101	100	11	
6	00010	0100	100	100	100	011	010	
7	000011	0011	011	0011	011	010	0001	
8	000010	0010	0010	011	0010	0001	001	
9	0000011	00011	00011	0010	00001	001	000000	
10	0000010	00010	00010	00010	0001	000000	-	
11	00000011	000011	000001	00001	00000	-	-	
12	00000010	000010	00001	00000	-	-	-	
13	000000011	000001	000000	-	-	-	-	
14	000000010	000000	-	-	-	-	-	
15	000000001	-	-	-	-	-	-	
TotalNumCoeff *TotalZeros*	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	000001	000001	00001	0000	0000	000	00	0
1	0001	000000	00000	0001	0001	001	01	1
2	00001	0001	001	001	01	1	1	-
3	011	11	11	010	1	01	-	-
4	11	10	10	1	001	-	-	-
5	10	001	01	011	-	-	-	-
6	010	01	0001	-	-	-	-	-
7	001	00001	-	-	-	-	-	-
8	000000	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-

Table 9-17 – TotalZeros table for chroma DC 2x2 blocks

NumCoeff TotalZeros	1	2	3
0	1	1	1
1	01	01	0
2	001	00	-
3	000	-	-

9.1.6.4.2 Run before each coefficient

~~At this stage it is known how many zeros are left to distribute (call this ZerosLeft). When decoding a non-zero coefficient for the first time, ZerosLeft begins at TotalZero, and decreases as more non-zero coefficients are decoded.~~

~~For example, if there is only 1 zero left, the run before the next coefficient must be either of length 0 or 1, and only one bit is needed.~~

~~The number of preceding zeros before each non-zero coefficient (called RunBefore) needs to be decoded to properly locate that coefficient. Before decoding the next RunBefore, ZerosLeft is updated and used to select one out of 7 tables. RunBefore does not need to be decoded in the following two situations:~~

- ~~If the total number of zeros has been reached (ZerosLeft = 0)~~
- ~~For the last coefficient in the backward scan. Then the value is known to be ZerosLeft. This also means that the maximum value to be coded is 14.~~

Formatted: Bullets and Numbering

At this stage it is known how many zeros are left to distribute (call this ZerosLeft). When decoding a run_before for the first time, ZerosLeft begins at TotalZeros, and decreases as more run_before elements are decoded.

For example, if there is only 1 zero left, the run before the next coefficient must be either of length 0 or 1, and only one bit is needed.

The number of preceding zeros before each non-zero coefficient (called RunBefore) needs to be decoded to properly locate that coefficient. Before decoding the next RunBefore, ZerosLeft is updated and used to select one out of 7 tables. RunBefore does not need to be decoded in the following two situations:

- If the total number of zeros has been reached (ZerosLeft = 0)
- For the last coefficient in the reverse zig-zag scan. Then the value is known to be ZerosLeft. This also means that the maximum value to be decoded is 14.

Formatted: Bullets and Numbering

Table 9-18 – Tables for run_before~~Tables for Run before each coefficient~~

TotalZeros / Run Before	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	001	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8		-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

9.2 Context-based adaptive binary arithmetic coding (CABAC)

9.2.1 Decoding flow and binarization

A binarization scheme provides a mapping of a non-binary valued alphabet of symbols onto a set of sequences of binary decisions, so-called bins. In subclauses 9.2.1.1 - 9.2.1.4 the elementary types of binarization schemes for CABAC are specified.

A specification of the decoding flow and the assignment of binarization schemes for all syntax elements is given in subclauses 9.2.1.5 - 9.2.1.9.

9.2.1.1 Unary binarization

Table 9-19 shows the first five codewords of the unary code used for binarization of code symbols. For a code symbol C it consists of $|C|$ '1' bits followed by the last bit with value '0'. The first bin number corresponds to the first bit of the unary codeword with increasing bin numbers towards the last bit, as shown in Table 9-19.

Table 9-19 – Binarization by means of the unary code tree

Code symbol	Binarization						
0	0						
1	1	0					
2	1	1	0				
3	1	1	1	0			
4	1	1	1	1	0		
5	1	1	1	1	1	0	
bin_num	1	2	3	4	5	6	

9.2.1.2 Truncated unary (TU) binarization

The truncated unary (TU) binarization is defined for a finite alphabet $\{0,\ldots,C_{max}\}$ by applying the unary binarization of subclause 9.2.1.1 to all code symbols C with $C < C_{max}$; the binarization of the symbol $C=C_{max}$ is defined by a code word consisting of C_{max} '1's (without the last bit of value'0'). Numbering of the bins is the same as for unary binarization.

9.2.1.3 Concatenated unary/ k^{th}-order Exp-Golomb (UEGk) binarization

Concatenated unary/k^{th}-order Exp-Golomb (UEGk) binarization consists of a concatenation of a prefix code word and a suffix code word. The prefix is formed by using a truncated unary binarization with $C_{max}=UCoff$, where $UCoff$ denotes the cut-off parameter. For all code symbols C with $C < UCoff$, the suffix code word is void; for code symbols C with $C \geq UCoff$, the suffix consists of an Exp-Golomb code of order k for the symbol $C-UCoff$. For a given symbol S the Exp-Golomb code of order k is constructed as follows:

```
while(1) {
    //first unary part of EGk
    if (symbol >= (unsigned int)(1<<k)) {
       put('1');
       S = S - (1<<k);
       k++;'
    }
    else
    {
       put('0');      //now terminating zero of unary part of EGk
       while (k--)    //finally binary part of EGk
          put( (S>>k) & 0x01 );
       break;
    }
}
```

The numbering of bins is such that bin number bin_num = 1 relates to the first binary decision in the unary prefix code with increasing numbers towards the least significant bits of the binary part of the Exp-Golomb suffix.

Formatted: Default Paragraph Font

Formatted: Default Paragraph Font

Formatted: Default Paragraph Font

9.2.1.4 Fixed-length (FL) binarization

Fixed-length (FL) binarization is applied to a finite alphabet $\{0, \ldots, C_{max}\}$ of code symbols. It is constructed by using a L-bit binary representation of a code symbol, where $L = \lfloor \log_2 C_{max} \rfloor + 1$. The numbering of bins for the fixed-length binarization is such that the bin_num = 1 relates to the least significant bit with increasing bin numbers towards the most significant bit.

Formatted: Default Paragraph Font

9.2.1.5 Binarization schemes for macroblock type and sub macroblock type

The binarization scheme for decoding of macroblock type in I slices is specified in Table 9-20. If adaptive_block_size_transform_flag == 1, the binarization for decoding of macroblock type in I slices is specified in Table 12-~~11~~10.

For macroblock types in SI slices the binarization consists of a prefix and a suffix part. The prefix consists of a single bit $b_1 = ((mb\_type == SIntra\_4x4) ? 0 : 1)$. The suffix part for mb_type Sintra_4x4 is void, while the suffix parts for all other macroblock types are given by the corresponding binarization pattern specified in Table 9-20.

The binarization schemes for P, SP, and B slices are specified in Table 9-21. The binarization for intra macroblock types in P and SP slices corresponding to mb_type values 7 to 30 consists of a prefix as specified in Table 9-21 and a suffix as specified in Table 9-20 for the corresponding intra mb_type. For intra macroblock types in B slices (mb_type values 23 to 47) the binarization consists of a prefix as specified in Table 9-21 and a suffix as specified in Table 9-20 for the corresponding intra mb_type. If adaptive_block_size_transform_flag == 1, the same prefix is used as specified in Table 9-21 for intra macroblock types of the corresponding slice type. However, the corresponding suffix parts are specified in Table 12-~~11~~10.

For P, SP, and B slices the specification of the binarization for sub_mb_type is given in Table 9-22.

Table 9-20 – Binarization for macroblock types for I slices

Value (name) of mb_type	Binarization					
0 (Intra_4x4)	0					
1 (Intra_16x16_0_0_0)	1	0	0	0	0	
2 (Intra_16x16_1_0_0)	1	0	0	0	1	
3 (Intra_16x16_2_0_0)	1	0	0	1	0	
4 (Intra_16x16_3_0_0)	1	0	0	1	1	
5 (Intra_16x16_0_1_0)	1	0	1	0	0	0
6 (Intra_16x16_1_1_0)	1	0	1	0	0	1
7 (Intra_16x16_2_1_0)	1	0	1	0	1	0
8 (Intra_16x16_3_1_0)	1	0	1	0	1	1
9 (Intra_16x16_0_2_0)	1	0	1	1	0	0
10 (Intra_16x16_1_2_0)	1	0	1	1	0	1
11 (Intra_16x16_2_2_0)	1	0	1	1	1	0
12 (Intra_16x16_3_2_0)	1	0	1	1	1	1
13 (Intra_16x16_0_0_1)	1	1	0	0	0	
14 (Intra_16x16_1_0_1)	1	1	0	0	1	
15 (Intra_16x16_2_0_1)	1	1	0	1	0	
16 (Intra_16x16_3_0_1)	1	1	0	1	1	
17 (Intra_16x16_0_1_1)	1	1	1	0	0	0
18 (Intra_16x16_1_1_1)	1	1	1	0	0	1
19 (Intra_16x16_2_1_1)	1	1	1	0	1	0

20 (Intra_16x16_3_1_1)	1	1	1	0	1	1
21 (Intra_16x16_0_2_1)	1	1	1	1	0	0
22 (Intra_16x16_1_2_1)	1	1	1	1	0	1
23 (Intra_16x16_2_2_1)	1	1	1	1	1	0
24 (Intra_16x16_3_2_1)	1	1	1	1	1	1
bin_num	1	2	3	4	5	6

Table 9-21 – Binarization for macroblock types for P, SP, and B slices

Slice type	Value (name) of mb_type	Binarization						
P, SP slices	0 (Pred_L0_16x16)	0	0	0				
	1 (Pred_L0_L0_16x8)	0	1	1				
	2 (Pred_L0_L0_8x16)	0	1	0				
	4 (Pred_8x8)	0	0	1				
	6 (Pred_8x8ref0)	na						
	7 to 30 (Intra, prefix only)	1						
B slices	0 (Direct 16x16)	0						
	1 (Pred_L0_16x16)	1	0	0				
	2 (BiPred_L1_16x16)	1	0	1				
	3 (BiPred_Bi_16x16)	1	1	0	0	0	0	
	4 (Pred_L0_L0_16x8)	1	1	0	0	0	1	
	5 (Pred_L0_L0_8x16)	1	1	0	0	1	0	
	6 (BiPred_L1_L1_16x8)	1	1	0	0	1	1	
	7 (BiPred_L1_L1_8x16)	1	1	0	1	0	0	
	8 (BiPred_L0_L1_16x8)	1	1	0	1	0	1	
	9 (BiPred_L0_L1_8x16)	1	1	0	1	1	0	
	10 (BiPred_L1_L0_16x8)	1	1	0	1	1	1	
	11 (BiPred_L1_L0_8x16)	1	1	1	1	1	0	
	12 (BiPred_L0_Bi_16x8)	1	1	1	0	0	0	0
	13 (BiPred_L0_Bi_8x16)	1	1	1	0	0	0	1
	14 (BiPred_L1_Bi_16x8)	1	1	1	0	0	1	0
	15 (BiPred_L1_Bi_8x16)	1	1	1	0	0	1	1
	16 (BiPred_Bi_L0_16x8)	1	1	1	0	1	0	0
	17 (BiPred_Bi_L0_8x16)	1	1	1	0	1	0	1
	18 (BiPred_Bi_L1_16x8)	1	1	1	0	1	1	0
	19 (BiPred_Bi_L1_8x16)	1	1	1	0	1	1	1
	20 (BiPred_Bi_Bi_16x8)	1	1	1	1	0	0	0
	21 (BiPred_Bi_Bi_8x16)	1	1	1	1	0	0	1

	22 (BiPred_8x8)	1	1	1	1	1	1	
	23 to 47 (Intra, prefix only)	1	1	1	1	0	1	
bin_num		1	2	3	4	5	6	7

Table 9-22 – Binarization for sub macroblock types in P and B slices

Slice type	Value (name) of sub_mb_type	Binarization					
P slices	0 (Pred_L0_8x8)	1					
	1 (Pred_L0_8x4)	0	0	0			
	2 (Pred_L0_4x8)	0	0	1	1		
	3 (Pred_L0_4x4)	0	0	1	0		
	4 (Intra_8x8)	0	1				
B slices	0 (Direct_8x8)	0					
	1 (Pred_L0_8x8)	1	0	0			
	2 (BiPred_L1_8x8)	1	0	1			
	3 (BiPred_Bi_8x8)	1	1	0	0	0	
	4 (Pred_L0_8x4)	1	1	0	0	1	
	5 (Pred_L0_4x8)	1	1	0	1	0	
	6 (BiPred_L1_8x4)	1	1	0	1	1	
	7 (BiPred_L1_4x8)	1	1	1	0	0	0
	8 (BiPred_Bi_8x4)	1	1	1	0	0	1
	9 (BiPred_Bi_4x8)	1	1	1	0	1	0
	10 (Pred_L0_4x4)	1	1	1	0	1	1
	11 (BiPred_L1_4x4)	1	1	1	1	0	0
	12 (BiPred_Bi_4x4)	1	1	1	1	0	1
	13 (Intra_8x8)	1	1	1	1	1	
bin_num		1	2	3	4	5	6

9.2.1.6 Decoding flow and assignment of binarization schemes

Formatted: Default Paragraph Font

In this subclause, the binarization schemes used for decoding of coded_block_pattern, delta_qp, the syntax elements of reference picture index, motion vector data, Intra4x4 prediction modes and are specified.

The coded block pattern information is decoded using the relationship as given in subclause 7.4.6: coded_block_pattern = coded_block_patternY + 16*nc. In a first step, the luma part coded_block_patternY of coded_block_pattern is decoded using the fixed-length (FL) binarization with C_{max} = 15 and L = 4. Then, the chroma part nc is decoded using TU binarization with C_{max} = 2.

Decoding of the delta_qp parameter is a two-step process. First, an unsigned code value *wrapped_delta_qp* ≥ 0 is decoded using the unary binarization. Then, *wrapped_delta_qp* is mapped to the signed value of the delta_qp parameter according to the relationship given in Table 9-2.

The decoding process for spatial intra prediction modes associated with luma of macroblock type Intra_4x4 and Sintra_4x4 is as follows. First, a parameter *intra_pred_indicator* is decoded using the truncated unary (TU) binarization with C_{max} = 8. If intra_pred_indicator = 0, use_most_probable_mode is set to 1. If intra_pred_indicator ≥ 1, remaining_mode_selector = intra_pred_indicator – 1. Given the parameters most_probable_mode and

remaining_mode_selector, the intra prediction mode intra_pred_mode is obtained in the same way as specified in subclause 9.1.5.1. The decoding order of the prediction modes is the same as shown in Figure 9-1 b). For decoding of the spatial intra prediction mode for chroma intra_chroma_pred_mode, the truncated unary (TU) binarization with $C_{max} = 3$ is used.

The reference picture index parameter is decoded using the unary binarization as given in subclause 9.2.1.1.

Each component of the motion vector data is decoded separately starting with the horizontal (h) component. First the absolute value *abs_mvd_comp* and then the sign *sign_mvd_comp* of each component (*comp=h,v*) shall be decoded. The binarization scheme applied to *abs_mvd_comp* is given by the concatenated unary/3rd-order Exp-Golomb (UEG3) binarization with cut-off parameter Ucoff = 9. Note that for decoding the Exp-Golomb suffix the decoder bypass Decode_eq_prob as specified in subclause 9.2.4.3.5 is used.

Formatted: Default Paragraph Font

9.2.1.7 Decoding flow and binarization of transform coefficients

Decoding of transform coefficients is a three-step process. First, the one-bit coded_block_flag is decoded for each block of transform coefficients unless the coded_block_pattern symbol on macroblock level indicates that the regarded block has no non-zero coefficients. If the coded_block_flag symbol is zero, no further information has to be decoded for the block. Otherwise, it is indicated that there are significant coefficients inside the block. The latter case implies that, in a second decoding step, for each scanning position i except the last position in a block the binary-valued significant_coeff_flag[i] has to be decoded. If significant_coeff_flag[i] has the value of one, the corresponding position i in the block contains a significant coefficient and a further binary-valued last_significant_coeff_flag[i] is decoded. If last_significant_coeff_flag[i] is zero, there is at least one further significant coefficient to be decoded; otherwise, the last significant coefficient along the scanning path is reached. If this is the case, the absolute value minus 1 coeff_absolute_value_minus_1 and then the sign of the coefficient coeff_sign is decoded for each significant transform coefficient by traversing the block in reverse scanning order. coeff_absolute_value_minus_1 is decoded using the concatenated unary/zero-order Exp-Golomb (UEG0) binarization with UCoff=14. Similar to the decoding of absolute values of the motion vector components, the Exp-Golomb suffix is decoded by using the decoder bypass Decode_eq_prob.

9.2.1.8 Decoding of sign information related to motion vector data and transform coefficients

Decoding of the sign information *sign_mvd_comp* of the motion vector components and *coeff_sign* of the levels corresponding to significant transform coefficients is performed as follows. Using the decoder bypass Decode_eq_prob as specified in subclause 9.2.4.3.5 first a binary indicator *sign_ind* is decoded. Then the sign information sign_info is recovered by *sign_info = ((sign_ind = = 0) ? 1 : -1)*.

9.2.1.9 Decoding of macroblock skip flag and end-of-slice flag

Decoding of the mb_skip_flag is as follows: First, a binary-valued *mb_skip_flag_decoded* is decoded using the context model as specified in subclause 9.2.2.2. In a second step, the actual value of mb_skip_flag is obtained by inverting mb_skip_flag_decoded, *i.e.*, mb_skip_flag = mb_skip_flag_decoded ^ 0x01.

The end_of_slice_flag is decoded using a fixed, non-adaptive model by chosing State = 63 and MPS = 0. The following mechanism guarantees a fixed model, although the coding engine uses a probability estimator after each decoding step as further specified in subclause 9.2.4.2. By observing a sequence of end_of_slice values '0' meaning that the end of a slice has not been reached, the initial chosen state will not be altered, since for the observation of a MPS symbol the state variable State = 63 will be mapped onto itself in the probability estimation. However, as soon as a LPS value of '1' is decoded for end_of_slice_flag, the probability estimation for the LPS will not affect the subsequent decoding process, because the end of a slice is reached, and all context models are refreshed using the initial states.

9.2.2 Context definition and assignment

For each bin number, a *context variable* is defined by a conditioning term containing prior decoded symbols or parts thereof. The possible numerical values of a context variable specify the different *context models* associated with a specific bin number. Typically, there are several possible values or *context labels* for each bin number bin_num. However, in some cases the context variable may simply be a constant label, in which case there is only one fixed context model.

This subclause defines first a variety of generic types of context variables, so-called *context templates*, for conditional coding of syntax elements. Then, for each bin number of a syntax element, the specification of the corresponding context variable is given. For the different bin numbers associated with the binarization of a given syntax element or parts thereof, a unique *context identifier* context_id is chosen such that the context variable associated to bin number k is given by context_id[k]. Since there is always a maximum bin number N with a context variable context_id[N] that is different from the corresponding context variable context_id[N−1] for the preceding bin number N−1, it is sufficient to specify a context identifier for each index k with $1 \le k \le N$, where N is called the maximum index of the context identifier *max_idx_ctx_id*.

Table 9-23 provides an overview of the context identifiers associated to each category of syntax elements. A detailed description of the corresponding context variables is given in the subsequent subclauses. Note that each context identifier corresponds to a unique range of context labels, which, in the case of macroblock type, may overlap for the different slice types I, SI, P, SP, and B.

If adaptive_block_size_transform_flag == 1, the context identifiers related to decoding of transform coefficients utilize an additional set of ranges as further specified in Table 12-12.

Table 9-23 – Syntax elements and associated context identifiers

Syntax element	Context identifier	Type of Binarization	max_idx_ctx_id	Range of context label
mb_skip_flag	ctx_mb_skip	-/-	1	0 – 2
mb_type	ctx_mb_type_I	Table 9-14	5	0 – 7
	ctx_mb_type_SI_pref	-/-	1	0 – 2
	ctx_mb_type_SI_suf	Table 9-14	5	3 – 10
	ctx_mb_type_P	Table 9-15	3	3 – 6
	ctx_mb_type_B		4	3 – 8
	ctx_mb_type_P_suf	Table 9-14	5	6 – 9
	ctx_mb_type_B_suf		5	8 – 11
	ctx_b8_mode_P	Table 9-16	4	12 – 15
	ctx_b8_mode_B		4	12 – 15
mvd_l0 and mvd_l1	ctx_abs_mvd_h	UEG3, UCoff=9	5	16 – 22
	ctx_abs_mvd_v	UEG3, UCoff=9	5	23 – 29
ref_idx_l0 and ref_idx_l1	ctx_ref_idx	Unary	3	30 – 35
delta_qp	ctx_delta_qp	Unary	3	36 – 39
chroma_pred_mode	ctx_ipred_chroma	TU, C_{max}=3	3	40 – 44
intra_pred_mode	ctx_ipred_luma	TU, C_{max}=8	2	45 – 62
coded_block_pattern	ctx_cbp_luma	FL, C_{max}=15	4	63 – 66
	ctx_cbp_chroma	TU, C_{max}=2	2	67 – 74
coded_block_flag	ctx_cbp4	-/-	-/-	75 – 94
significant_coeff	ctx_sig	-/-	-/-	95 – 155
last_coeff	ctx_last	-/-	-/-	156 – 216
coeff_absolute_value_minus_1	ctx_abs_level	UEG0, UCoff=14	2	217 – 266
end_of_slice_flag	ctx_eos	Fixed, non-adaptive model with State(ctx_eos) = 63, MPS(ctx_eos) = 0		

Formatted Table

Formatted: Normal

9.2.2.1 Overview of assignment of context labels

Tables 9-24 and 9-25 contain all context identifiers along with their corresponding range of context labels. The association of context labels (modulo some offset) and bin numbers shows which context variable uses a fixed model and which one implies a choice of different models. The latter are characterized by those entries where a set of different context labels are given for a specific bin number bin_num (Table 9-24) or block type dependent context_category (Table 9-25). These context variables are specified in the following subclauses.

Table 9-24 – Overview of context identifiers and associated context labels

Context identifier	Range of context label	Offset for context label	max_idx_ctx_id	bin_num				
				1	2	3	4	≥ 5
ctx_mb_skip	0 – 2	0	1	0,1,2	-/-	-/-	-/-	-/-
ctx_mb_type_I	0 – 7	0	5	0,1,2	3	4	5,6	6,7
ctx_mb_type_SI_pref	0 – 2	0	1	0,1,2	-/-	-/-	-/-	-/-
ctx_mb_type_SI_suf	3 – 10	3	5	0,1,2	3	4	5,6	6,7
ctx_mb_type_P	3 – 6	3	3	0	1	2,3	-/-	-/-
ctx_mb_type_P_suf	6 – 9	6	5	0	1	2	2,3	3
ctx_mb_type_B	3 – 8	3	4	0,1,2	3	4,5	5	5
ctx_mb_type_B_suf	8 – 11	8	5	0	1	2	2,3	3
ctx_b8_mode_P	12 – 15	12	4	0	1	2	3	-/-
ctx_b8_mode_B	12 – 15	12	4	0	1	2,3	3	3
ctx_abs_mvd_h	16 – 22	16	5	0,1,2	3	4	5	6
ctx_abs_mvd_v	23 – 29	23	5	0,1,2	3	4	5	6
ctx_ref_idx	30 – 35	30	3	0,1,2,3	4	5	5	5
ctx_delta_qp	36 – 39	36	3	0,1	2	3	3	3
ctx_ipred_chroma	40 – 44	36	3	0,1,2	3	4	-/-	-/-
ctx_ipred_luma	45 – 62	42	2	0,...,8	9,...,17	9,...,17	9,...,17	9,...,17
ctx_cbp_luma	63 – 66	60	4	0,1,2,3	0,1,2,3	0,1,2,3	0,1,2,3	-/-
ctx_cbp_chroma	67 – 74	64	2	0,1,2,3	4,5,6,7	-/-	-/-	-/-
ctx_abs_level	217 – 266	217+ 10*context_category	2	0,...,4	5,...,9	5,...,9	5,...,9	5,...,9

Table 9-25 – Overview of context identifiers and associated context labels (continued)

Context identifier	Offset (range) of context label	context_category (as specified in Table 9-22)				
		0	1	2	3	4
ctx_cbp4	75 (75 – 94)	0 – 3	4 – 7	8 – 11	12 – 15	16 – 19
ctx_sig	95 (95 – 155)	0 – 14	15 – 28	29 – 43	44 –46	47 – 60
ctx_last	156 (156 – 216)	0 – 14	15 – 28	29 – 43	44 –46	47 – 60

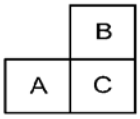

Figure 9-2 – Illustration of the generic context template using two neighbouring symbols A and B for conditional coding of a current symbol C

9.2.2.2 Context templates using two neighbouring symbols

The generic design of this type of context variable is shown in Figure 9-2. It involves two previously decoded symbols or bins of the same syntax element that correspond to the spatially neighbouring blocks to the left (A) and on the top (B) of the regarded block C. The generic form of the equation defining this type of context is given by

$$ctx\_var\_spat = cond\_term(A, B), \tag{9-1}$$

where the conditioning term *cond_term(A, B)* describes the functional relationship between the spatially neighbouring symbols *A* and *B*, on the one hand, and the values of the context variable, on the other hand. Three special cases of this template are specified as follows:

$$ctx\_var\_spat1 = cond\_term(A) + cond\_term(B), \tag{9-2}$$

$$ctx\_var\_spat2 = cond\_term(A) + 2*cond\_term(B), \tag{9-3}$$

$$ctx\_var\_spat3 = cond\_term(A). \tag{9-4}$$

Table 9-26 contains the specification of context variables relying on templates using two neighbouring symbols. The conditioning term of the context variable *ctx_cbp4* depends on six block types as given in Table 9-28 (Luma-DC, Luma-AC, Chroma-U-DC, Chroma-V-DC, Chroma-U-AC, Chroma-V-AC).

Table 9-26 – Specification of context variables using context templates according to Equations (9-2) – (9-4)

Context variable	Context template	cond_term(X), semantics of X	cond_term(X), if X not available
ctx_mb_skip	*ctx_var_spat1*	(mb_skip_flag(X) == 0) ? 1: 0 X: neighbouring macroblock	0
ctx_mb_type_I[1]	*ctx_var_spat1*	(mb_type(X) != Intra_4x4) ? 1: 0 X: neighbouring macroblock	0
ctx_mb_type_SI_pref[1]	*ctx_var_spat1*	(mb_type(X) != Sintra_4x4) ? 1: 0 X: neighbouring macroblock	0
ctx_mb_type_SI_suf[1]	*ctx_var_spat1*	(mb_type(X) != Intra_4x4) ? 1: 0 X: neighbouring macroblock	0
ctx_mb_type_B[1]	*ctx_var_spat1*	((mb_type(X) != Direct) ? 1: 0) X: neighbouring macroblock	0
ctx_ipred_chroma[1]	*ctx_var_spat1*	((intra_chroma_pred_mode(X) != 0) ? 1: 0) X: neighbouring macroblock	0
ctx_ref_idx[1]	*ctx_var_spat2*	(ref_idx_l0/ref_idx_l1(X) != 0) ? 1: 0 X: neighbouring block	0
ctx_ipred_luma[i], i=1,2	*ctx_var_spat3*	9*(i–1) + intra_pred_mode(X) X: neighbouring block	0
ctx_cbp_luma[i], i=1,…,4	*ctx_var_spat2*	i-th bit of coded_block_patternY(X) X: neighbouring 8x8 block of i-th block	0
ctx_cbp_chroma[1]	*ctx_var_spat2*	(nc(X) != 0) ? 1: 0 X: neighbouring macroblock	0

ctx_cbp_chroma[2]	*ctx_var_spat2*	4 + (nc(X) == 2) ? 1: 0 X: neighbouring macroblock	0
ctx_cbp4	*ctx_var_spat2*	coded_block_pattern_4(X) X: neighbouring block of same block type	1, if X is INTRA; 0, otherwise
ctx_delta_qp	*ctx_var_spat3*	(delta_qp(X) != 0) ? 1: 0 X: neighbouring macroblock	0

The specification of the context variables ctx_abs_mvd_h[1] and ctx_abs_mvd_v[1] is given as follows.

$$ctx\_abs\_mvd\_comp[1] = \begin{cases} 0, (|mvd\_comp(A)| + |mvd\_comp(B)|) < 3, \\ 2, (|mvd\_comp(A)| + |mvd\_comp(B)|) > 32, \\ 1, otherwise, \end{cases} \quad (9\text{-}5)$$

where *comp* denotes the component h (horizontal) or v (vertical) and *A*, *B* denote the neighbouring blocks of the block to decode as shown in Figure 9-2. Since the neighbouring blocks *A* and *B* may belong to different macroblock partitions, the following principle for identifying the proper neighbouring blocks that are used in Equation (9-5) is established. First, the motion vector data is assumed to be given in oversampled form such that each 4x4 block has its own *motion vector (MV)*. That means, on the one hand, that in case of a neighbouring block having a coarser partition, the related 4x4 sub-blocks are assumed to inherit the *MV* from the corresponding parent block(s) in the quadtree partition. On the other hand, if the current block *C* represents a larger block than a 4x4 block, it is assumed to be represented by the corresponding leftmost 4x4 sub-block in the top row of 4x4 sub-blocks. Then, given a block *C*, the neighbouring 4x4 sub-blocks *B* on top and *A* to the left of the representing 4x4 sub-block of *C* are chosen for evaluation of Equation (9-5).

9.2.2.3 Context templates using preceding bin values

Let (b_1, …, b_N) denote the binarization of a given symbol C. Then, a generic type of context variable associated with the k-th bin of C is specified as

$$ctx\_var\_bin[k] = cond\_term(b_1, \ldots, b_{k-1}), \quad (9\text{-}6)$$

where $1<k\leq N$. In Table 9-27, the specification of context variables using this type of context template is given.

Table 9-27 – Definition of context variables using the context template according to Equation (9-6)

Context variable	cond_term(b_1, …, b_{k-1})
ctx_mb_type_I[4]	(b_3 == 0) ? 5: 6
ctx_mb_type_I[5]	(b_3 == 0) ? 6: 7
ctx_mb_type_SI_suf[4]	(b_3 == 0) ? 5: 6
ctx_mb_type_SI_suf[5]	(b_3 == 0) ? 6: 7
ctx_mb_type_P[3]	(b_2 == 0) ? 2: 3
ctx_mb_type_P_suf[4]	(b_3 == 0) ? 2: 3
ctx_mb_type_B[3]	(b_2 == 1) ? 4: 5
ctx_mb_type_B_suf[4]	(b_3 == 0) ? 2: 3
ctx_b8_mode_B[3]	(b_2 == 1) ? 2: 3

9.2.2.4 Additional context definitions for information related to transform coefficients

Three different additional context identifiers are used for conditioning of information related to transform coefficients. All these three types depend on context categories of different block types denoted by the variable *context_category*. The definition of these context categories is given in Table 9-28.

Table 9-28 – Context categories for the different block types

block_type	MaxNumCoeff	context_category
Luma DC block for INTRA16x16 mode	16	0:Luma-Intra16-DC
Luma AC block for INTRA16x16 mode	15	1:Luma-Intra16-AC
Luma block for INTRA 4x4 mode	16	2:Luma-4x4
Luma block for INTER 4x4 mode	16	
U-Chroma DC block for INTRA mode	4	3:Chroma-DC
V-Chroma DC block for INTRA mode	4	
U-Chroma DC block for INTER mode	4	
V-Chroma DC block for INTER mode	4	
U-Chroma AC block for INTRA mode	15	4:Chroma-AC
V-Chroma AC block for INTRA mode	15	
U-Chroma AC block for INTER mode	15	
V-Chroma AC block for INTER mode	15	

Additional context categories are used in the case of adaptive_block_size_transform_flag == 1 as specified in subclause 12.5.2. The context identifiers ctx_sig and ctx_last are related to the binary valued information of SIG and LAST; the related context variables includes an additional dependency on the scanning position *scanning_pos* within the regarded block:

$$ctx\_sig[scanning\_pos] = Map\_sig(scanning\_pos), \quad (9\text{-}7)$$

$$ctx\_last[scanning\_pos] = Map\_last(scanning\_pos). \quad (9\text{-}8)$$

The specifiaction of Map_sig and Map_last in Equations (9-7) and (9-8) depends on the block type. For context_category 0 – 4 the corresponding maps are given by the identity, i.e.,

$$Map\_sig(scanning\_pos) = Map\_last(scanning\_pos) = scanning\_pos, \; if \; context\_category = 0, \ldots, 4,$$

where scanning_pos denotes the position related to the zig-zag scan. For the additional context categories 5 – 7, which are only in use if adaptive_block_size_transform_flag == 1, the specification of Map_sig and Map_last is given in subclause 12.5.2.

For decoding of abs_level_m1, the absolute values of significant transform coefficients minus 1, the context identifier ctx_abs_level consisting of the two context variables ctx_abs_level[1] and ctx_abs_level[2] is used, which are defined as follows:

$$ctx\_abs\_lev[1] = ((num\_decod\_abs\_lev\_gt1!=0) \; ? \; 4 : min(3, num\_decod\_abs\_lev\_eq1)), \quad (9\text{-}9)$$

$$ctx\_abs\_lev[2] = min(4, \; num\_decod\_abs\_lev\_gt1), \quad (9\text{-}10)$$

where *num_decod_abs_lev_eq1* denotes the number of decoded coefficients with magnitude equal to 1, and *num_decod_abs_lev_gt1* denotes the number of decoded coefficients with magnitude greater than 1. Both numbers are related to the same transform coefficient block, where the current decoding takes place, which means that no additionalinformation outside the regarded transform coefficient block is used for the context variables ctx_abs_level[k], k=1,2.

9.2.3 Initialisation of context models

9.2.3.1 Initialisation procedure

At the beginning of each slice, each context model is initialised with an initial state, which consists of a state number and the meaning of the most probable symbol (MPS) as further described in subclause 9.2.4.2. The actual initial state of a

context model depends linearly on the (initial) quantization parameter QP of the given slice, such that for each context model a pair of table entries {m, n} is given, from which the initial state is computed in the following way

1. Compute pre_state = ((m*(QP−12))>>4) + n;
2. Limit pre_state to the range of [0,101] for P- and B-slices and to the range of [27,74] for I-slices, i.e., pre_state = min (101, max(0,pre_state)) for P- and B-slices and pre_state = min (74, max(27,pre_state)) for I-slices;
3. Map pre_state to {state, MPS} pair according to the following rule: if (pre_state <= 50) then {state = 50-pre_state, MPS = 0} else {state = pre_state-51, MPS = 1}

9.2.3.2 Initialisation procedure

Tables 9-29 – 9-34 contain the initialisation parameters related to the context variables of all syntax elements, from which the initial states can be obtained by using the rules given in subclause 9.2.3.1.

Table 9-29 – Initialisation parameters for context identifiers *ctx_mb_type_I, ctx_mb_type_SI_pref, , ctx_mb_type_SI_suf,ctx_mb_skip, ctx_mb_type_P, ctx_mb_type_B*

Context label	ctx_mb_type_I		ctx_mb_type_SI_pref		ctx_mb_skip					
	m	n	m	n	m	n	m	n	m	n
0	7	25	7	25	-23	66				
1	8	35	8	35	-14	77				
2	-2	63	-2	63	-9	88				
			ctx_mb_type_SI_suf				ctx_mb_type_P		ctx_mb_type_B	
3	-9	68	7	25			2	13	9	49
4	-15	74	8	35			14	24	3	65
5	-3	36	-2	63			-21	69	0	78
6	-1	51	-9	68			-1	52	-13	81
7	0	50	-15	74					-14	73
8			-3	36					-8	64
9			-1	51						
10			0	50						

Table 9-30 – Initialisation parameters for context identifiers *ctx_b8_mode_P, ctx_b8_mode_B, ctx_mb_type_P_suf, ctx_mb_type_B_suf*

Context label	ctx_mb_type_P_suf ctx_mb_type_B_suf		Context label	ctx_b8_mode_P		ctx_b8_mode_B	
	m	n		m	n	m	n
9	-9	55	12	8	46	-9	62
10	-7	50	13	12	11	-12	66
11	2	47	14	-4	62	-9	56
			15	18	48	3	47

Table 9-31 – Initialisation parameters for context identifiers *ctx_abs_mvd_h, ctx_abs_mvd_v, ctx_ref_idx*

Context label	ctx_abs_mvd_h		Context label	ctx_abs_mvd_v		Context label	ctx_ref_idx	
	m	n		m	n		m	n
16	1	48	23	-1	45	30	3	27
17	-5	60	24	-5	59	31	-1	47
18	-8	70	25	-9	71	32	0	45
19	2	52	26	0	50	33	-2	60
20	2	62	27	3	61	34	-1	57
21	-3	64	28	-3	63	35	0	48
22	-2	80	29	1	80			

Table 9-32 – Initialisation parameters for context identifiers *ctx_delta_qp, ctx_ipred_chroma, ctx_ipred_luma*

Context label	ctx_delta_qp		Context label	ctx_ipred_luma		Context label	ctx_ipred_luma	
	m	n		m	n		m	n
36	0	28	45	-5	49	54	-4	61
37	0	50	46	-3	58	55	-6	64
38	0	50	47	-5	58	56	-6	63
39	0	50	48	-4	58	57	-3	75
	ctx_ipred_chroma		49	-5	59	58	-4	63
40	-5	50	50	-4	60	59	-7	65
41	0	50	51	-6	61	60	-1	63
42	0	50	52	-5	62	61	-18	66
43	0	50	53	-4	60	62	-9	67
44	0	50						

Table 9-33 – Initialisation parameters for context identifiers *ctx_cbp_luma, ctx_cbp_chroma*

Context label	ctx_cbp_luma				Context label	ctx_cbp_chroma				Context label	ctx_cbp_chroma			
	I slices		P, B slices			I slices		P, B slices			I slices		P, B slices	
	m	n	m	n		m	n	m	n		m	n	m	n
63	-3	75	-21	81	67	-7	65	-23	58	71	-18	66	-11	46
64	-4	63	-15	60	68	-1	63	-18	64	72	-9	67	-6	56
65	-4	70	-14	61	69	-9	77	-16	63	73	-13	70	-8	59
66	-5	56	-15	47	70	-4	76	-18	73	74	-7	74	-18	74

Table 9-34 – Initialisation parameters for context identifiers *ctx_cbp4, ctx_sig, ctx_last, ctx_abs_level* for context category 0 – 4

Context label	Context category 0	Context label	Context category 1	Context label	Context category 2		Context label	Context category 3	Context label	Context category 4
					I slices	P, B slices				

	m	n		m	n		m	n	m	n		m	n		m	n
ctx_cbp4																
75	-4	72	79	-4	37	83	-2	52	-4	57	87	0	55	91	-3	41
76	-4	68	80	-3	50	84	-7	68	-9	66	88	-3	70	92	-4	62
77	-6	75	81	-6	49	85	-5	61	-7	64	89	-4	68	93	-6	58
78	-6	75	82	-3	61	86	-8	77	-17	80	90	-4	75	94	-8	73
ctx_sig																
95	-6	68				124	-7	67	4	46	139	-3	71			
96	-11	66	110	0	44	125	-11	64	1	43	140	-9	69	142	-6	60
97	-5	63	111	2	53	126	-8	66	2	45	141	0	70	143	0	63
98	-5	56	112	0	49	127	-11	63	-4	46				144	-3	54
99	2	43	113	1	43	128	-9	63	-1	45				145	-4	54
100	1	47	114	4	45	129	-8	60	3	43				146	4	52
101	-8	58	115	-2	40	130	-11	55	-6	44				147	-5	44
102	-3	46	116	-1	45	131	-10	61	1	45				148	-1	48
103	4	38	117	0	50	132	-7	63	2	46				149	-7	57
104	0	58	118	2	55	133	-7	60	0	46				150	11	51
105	1	51	119	-7	52	134	-16	61	-12	49				151	-13	51
106	-7	57	120	-2	57	135	-2	62	3	50				152	7	55
107	-1	53	121	7	50	136	-1	58	11	46				153	5	57
108	2	47	122	2	52	137	-8	61	4	50				154	2	51
109	-1	59	123	4	66	138	-3	68	9	64				155	4	68
ctx_last																
156	0	5				185	11	25	16	27	200	12	27			
157	1	1	171	4	42	186	9	24	21	19	201	12	28	203	10	28
158	2	2	172	5	46	187	12	24	20	23	202	16	38	204	14	30
159	3	6	173	9	40	188	14	23	21	22				205	17	30
160	4	3	174	7	41	189	13	23	21	23				206	20	30
161	5	4	175	6	46	190	16	22	24	23				207	15	37
162	6	4	176	10	40	191	19	20	25	24				208	21	39
163	7	3	177	14	33	192	18	21	23	27				209	22	33
164	8	4	178	10	43	193	21	21	25	29				210	21	39
165	9	5	179	12	48	194	23	25	23	35				211	15	52
166	10	9	180	13	39	195	20	23	19	36				212	8	49
167	11	2	181	13	41	196	24	25	21	40				213	13	52
168	12	3	182	16	43	197	25	29	23	45				214	8	60

169	13	1	183	21	35	198	24	33	15	53				215	15	56
170	14	6	184	11	55	199	14	53	8	70				216	3	71
ctx_abs_level																
217	-5	55	227	-10	57	237	-10	63	-6	51	247	-9	70	257	-7	58
218	-3	36	228	-1	30	238	-5	37	-5	24	248	-14	55	258	0	33
219	-1	35	229	0	32	239	-7	43	-7	32	249	-10	57	259	-1	40
220	-2	40	230	-1	35	240	-6	46	-4	34	250	-5	56	260	-2	45
221	-6	50	231	0	40	241	-5	49	-5	39	251	-4	57	261	-3	49
222	-3	44	232	-5	39	242	-8	50	-12	43	252	-14	63	262	-7	48
223	-4	51	233	-4	47	243	-7	56	-7	50	253	-11	67	263	-9	58
224	-3	53	234	-9	55	244	-9	62	0	48	254	-5	68	264	-16	66
225	-4	55	235	-6	58	245	-9	64	-3	53	255	-9	71	265	-12	65
226	-11	63	236	-4	56	246	-11	70	-8	60	256	0	50	266	-12	68

9.2.4 Table-based arithmetic coding

NOTE - Arithmetic coding is based on the principle of recursive interval subdivision. Given a probability estimation $p(\text{'0'})$ and $p(\text{'1'})=1-p(\text{'0'})$ of a binary decision ('0', '1'), an initially given interval with range R will be subdivided into two sub-intervals having range $p(\text{'0'})\times R$ and $R-p(\text{'0'})\times R$, respectively. Depending on the decision, which has been observed, the corresponding sub-interval will be chosen as the new code interval, and a binary code string pointing into that interval will represent the sequence of observed binary decisions. It is useful to distinguish between the *most probable symbol* (MPS) and the *least probable symbol* (LPS), so that binary decisions have to be identified as either MPS or LPS, rather than '0' or '1'. Given this terminology, each context model *CTX* is defined by the probability p_{LPS} of the LPS and the value of MPS, which is either '0' or '1'.

The arithmetic core engine in this Recommendation | International Standard has three distinct properties:

- The probability estimation is performed by means of a finite-state machine with a table-based transition process between 64 different representative probability states $\{P_k \mid 0 \le k < 64\}$ for the LPS probability p_{LPS}.
- The range R representing the state of the coding engine is quantized to a small set $\{Q_1,\ldots,Q_4\}$ of pre-defined quantization values prior to the calculation of the new interval range. Storing a table containing all 64×4 pre-computed product values of $Q_i \times P_k$ allows a multiplication-free approximation of the product $R \times P_k$.
- For syntax elements or parts thereof with an approximately uniform probability distribution a separate simplified encoding and decoding path is used.

9.2.4.2 Probability estimation

The probability estimator is realized by a finite-state machine (FSM) consisting of a set of representative probabilities $\{P_k \mid 0 \le k < 64\}$ for the LPS together with some appropriately defined state transition rules. Table 9-35 shows the transition rules for adapting to a given MPS or LPS decision. For transition from one state to another each state is only addressed by its index *State*, which will be appropriately changed to a new index *Next_State_MPS(State)* or *Next_State_LPS(State)* after the encoding/decoding of a *MPS* or *LPS* symbol, respectively.

The numbering of the states is arranged in such a way that the state with index *State*=0 corresponds to a LPS probability value of 0.5, with decreasing LPS probability towards higher states. However, for I-slices it is of advantage to restrict the number of states to the first 24 state indices. Therefore, Table 9-35 contains a separate column containing the transition rule *Next_State_MPS_INTRA* that is used for decoding the syntax elements of an I-slice only. Note, that *Next_State_MPS_INTRA* differs from *Next_State_MPS* only by one entry. To prevent the FSM from switching to states higher than *State*=23, we set *Next_State_MPS(35)*= 23 for I-slice decoding. For the clarity of presentation, a separate table entry for I-slice decoding is shown in Table 9-35.

After encoding or decoding a decision, an update of the probability estimate is obtained by switching the state index *State* to a new index, such that for I-slice coding

```
if(decision = = MPS)
        State ← Next_State_MPS_INTRA(State)
else
        State ← Next_State_LPS(State)
```

and all other slice types

```
if(decision = = MPS)
        State ← Next_State_MPS(State)
```